AD-A253 783

2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE June 30, 1992	3. REPORT TYPE AND DATES COVERED Quarterly Technical Report, 4/92 - 6/92

4. TITLE AND SUBTITLE

An MCM/Chip Concurrent Engineering Validation
1992 Second Quarter Technical Report

5. FUNDING NUMBERS

MDA972-92-C-0022

6. AUTHOR(S)

Dr. Hector Moreno

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Dr.
Austin, TX 78759

8. PERFORMING ORGANIZATION REPORT NUMBER

HVE-171-92

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA/DSO, Dr. H. Lee Buchanan
3701 N. Fairfax Dr.
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

DTIC
S ELECTE D
JUL 2 9 1992
C

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

No restrictions

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A schema to model the information required to do physical design of Multi-Chip Modules was written in the EXPRESS language and mapped into ROSE C++ classes. Software was written in the SKILL language (from Cadence Design Systems) to read and write from/to the Cadence EDGE layout system. C++ code was written to interface the EDGE system to the ROSE C++ classes referred to above. As a resukt there exists now a full path in and out of the EDGE system and the information stored by ROSE.

14. SUBJECT TERMS

Concurrent Engineering, Multi Chip Module, Computer Aided Design

			15. NUMBER OF PAGES 57
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

92 7 27 [illegible]

92-20285

MCC TECHNICAL REPORT
HVE-171-92

An MCM/Chip Concurrent Engineering Validation, 1992 2nd Quarter Report

MCC

Microelectronics and Computer Technology Corporation

An MCM/Chip Concurrent Engineering Validation

1992 Second Quarter Technical Report

Dr. Hector Moreno
High Value Electronics Division
Austin, TX 78727
June 30, 1992

Sponsored by
Defense Advanced Research Projects Agency
Defense Sciences Office
DARPA Initiative on Concurrent Engineering
ARPA Order No. 8363/07
Issued by DARPA/CMO under Contract #MDA972-92-C-0022

Trademarks

Table of Contents

Summary

This report presents the technical progress of the effort on Phase 1 of a project at the Microelectronics and Computer Technology Corporation (MCC) supported by the Defense Advanced Research Projects Agency (DARPA).

The principal project objective is to create and validate a concurrent engineering environment for the design of Multi-Chip Modules (MCMs).

On Phase 1, the focus is on physical design of MCMs. The objective in this phase is to integrate four different MCM layout systems: EDGE, Finesse, Allegro 6.0 and AutoCad. The principal technical problem addressed in this period was the integration of the EDGE layout system from Cadence Design Systems to the ROSE concurrent engineering object-oriented data manager. This problem was solved through the development of a software system written in three languages: EXPRESS, SKILL and C++.

EXPRESS was utilized to state the information model that ROSE implements and manages. SKILL is supported by the EDGE system and was utilized to read and write to the EDGE internal database. C++ was utilized to link the ROSE and the SKILL data.

In the course of this work we have determined an efficient method to make the total system work, and a strategy for saving and updating information. We plan to use these method and strategy in the subsequent integration tasks.

1. Project Objectives

1.1. Phase 1

- To create a concurrent engineering environment suitable for the physical design of Multi-Chip Modules (MCMs) by integrating several different physical layout Computer-Aided Design (CAD) tools through data sharing.
- To validate the use of ROSE as a concurrent engineering data manager .
- To validate the concurrent engineering approach to MCM design.
- To utilize the concurrent engineering environment to design MCMs.

1.2. Phase 2

Repeat the Phase 1 objectives but extending them individually to each of the following areas:

- MCM modeling.
- MCM simulation.
- Design for testability.

This phase will begin upon funding release.

2. Progress Overview

2.1. Technical progress achieved

During the months of January through June of 1992, the following work was done:

The ROSE software was acquired and installed. Two MCC project team members were trained at the STEP Tools, Inc. location in Troy, NY. Actual installation of the software at MCC took place in February, 1992.

Once ROSE was installed, the rest of the team members had access to it and proceeded to take the tutorial course that came with it in order to become proficient in its use.

Considering that the application we have in mind is to create the concurrent engineering environment suitable for the physical layout of MCMs the next step was to develop a layout information model and to implement it through a schema in order to store and retrieve the required data to achieve the successful design of an MCM.

Following this, we implemented the integration between ROSE and a commercial CAD layout package, EDGE 2.1., which is a product from Cadence Design Systems, Inc.

We have started integration work between Finesse. (an MCM layout package from Harris Electronic Design Automation) and ROSE, and also between Allegro 6.0 (a product from Cadence Design Systems, Inc.) and ROSE.

3. Technical Problems

3.1. Problems addressed

During the period covered by this review, we have focused exclusively on Phase 1 objectives. The principal technical problems that have been addressed are the following:

- The definition of the entities that are necessary for the description of the MCM layout data, taking into account that there will be a number of designers, each using one of a variety of heterogeneous design systems who will need read/write access to it. These entities are the building blocks from which the information model is built.
- To determine the overall architecture of the concurrent engineering environment.
- To determine the process and practice through which the actual concurrent design activity will take place.
- To implement pertinent solutions to these problems.

4. General Methodology

4.1. Layout Information Model

In order to define the entities needed to represent the MCM layout information it is necessary to understand the information which the individual CAD systems keep and which is required and needed for their proper utilization. Then one needs to abstract and deal first with that subset which is common to all of them and use it as a nucleus or core, from which one can expand and tailor to include what will be required in the concurrent use of the integrated systems.

In our case, there are two basic types of entities :

- Geometric information entities.
- Connectivity information entities.

Our method has been to examine what data types are required by a system which is predominantly used for layout, which we expect to be common among all the systems to be integrated and then to find which are necessary to complete the picture for systems with connectivity intelligence. Then each of these identified types is defined as an abstract entity. The information content is expressed through the interrelationship among all the entities. This interrelationship is known as the schema, which we codify through the EXPRESS language. The geometric entities we have included up to date are:

- Cells
- Arrays
- Cell Instances
- Geometric Primitives:
  - Paths
  - Ellipses
  - Rectangles

- Donuts
- Polygons
- Lines
- Points

The connectivity entities up to date are:

- Element-Pin pairs (elPin)
- Net
- Netlist

The schema that implements these entities is listed as Appendix A of this report.

4.2. Concurrent Engineering Layout System Architecture

At present, the basic architecture of the system is represented by the following figure:

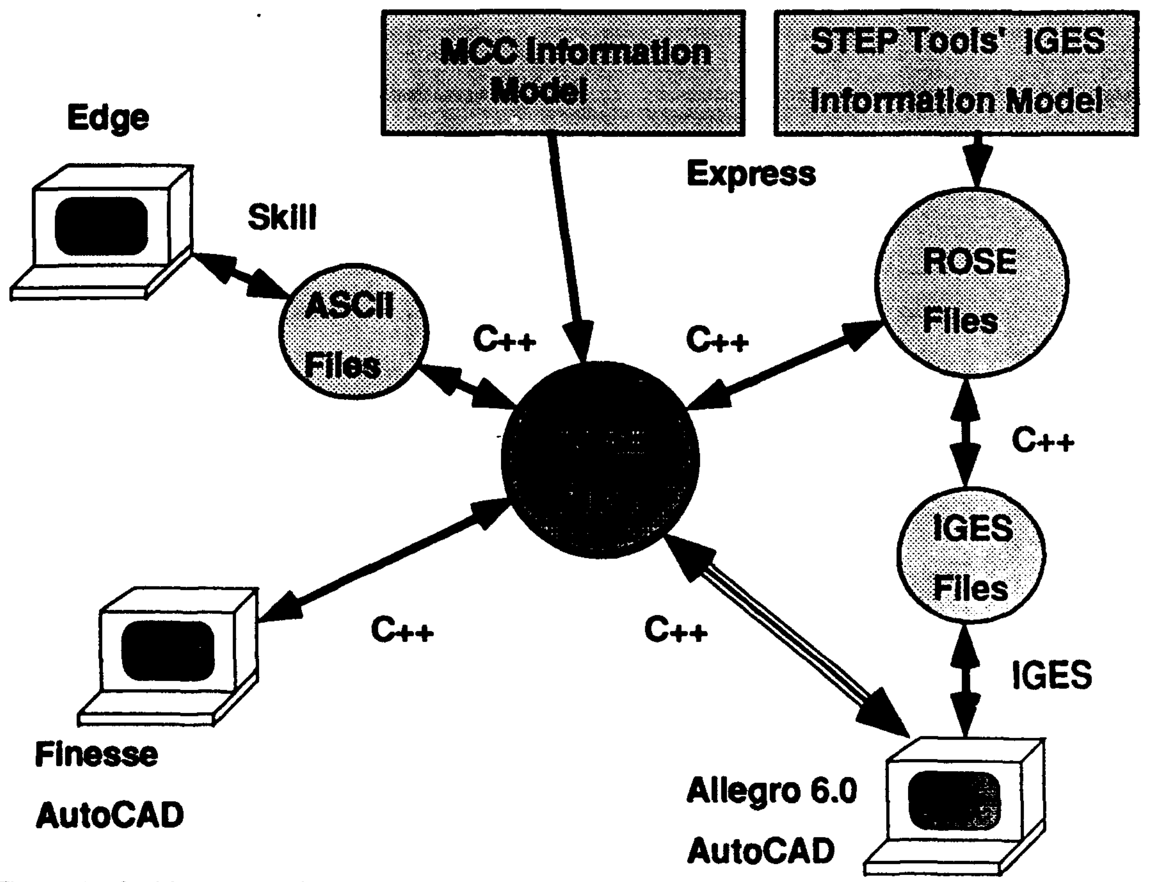

Figure 1. Architecture of Concurrent Engineering Design System.

Figure 1 shows the four types of physical design CAD systems we will be working with in Phase 1, and two ROSE implementations for two different information models: the one introduced in the previous section and the other, one that implements the Initial Graphics Exchange Standard (IGES) which is supported by STEP Tools, Inc.. IGES is a very general (3-dimensional) standard to represent graphical information and quite cumbersome for our purposes [NCGA]. However, we find it necessary to work with it because systems such as Allegro 6.0 accept IGES representation of graphical data as their input, while not providing many other acceptable means to reliably input data from external sources, nor a procedural language to read or write to its internal data.

Allegro 6.0 is capable of writing out its layout design data in IGES form.

AutoCad is able to read/write IGES files as well, so once the link is achieved for Allegro 6.0, we expect AutoCad will also be integrated to the system in the same form. In addition, it may be possible in the future to implement a more direct link between the MCC information model and AutoCad making use of direct C access to AutoCad's database, and this possibility is incorporated into Figure 1 by the presence of a direct double-arrow link.

The two information models are written in the EXPRESS language. There are software tools form STEP Tools, Inc. that compile those models into C++ classes. ROSE then provides many methods to create the class instances, to relate them according to the schema and to manage them. In addition, it is possible to map the ROSE objects created under one model to the ones created under the other through C++ ROSE code, and in that manner the integration will occur.

The link between Edge and ROSE is achieved through an intermediate text form. Data goes in and out of EDGE via software written in Cadence's SKILL language, which allows read/write access to EDGE's internal data structures. The link to ROSE is completed through a parser written in C++, which reads the intermediate form and creates the (C++) objects that implement the layout information model we introduced in the previous section. In the opposite direction, C++ software navigates through the ROSE objects and creates the intermediate form description.

For the Finesse software from Harris we expect it will be possible to make a direct link between ROSE and it, since Harris owns the Finesse source code.

4.3. Concurrent Engineering Layout System Operation

The operation of the system is visualized through Figure 2.

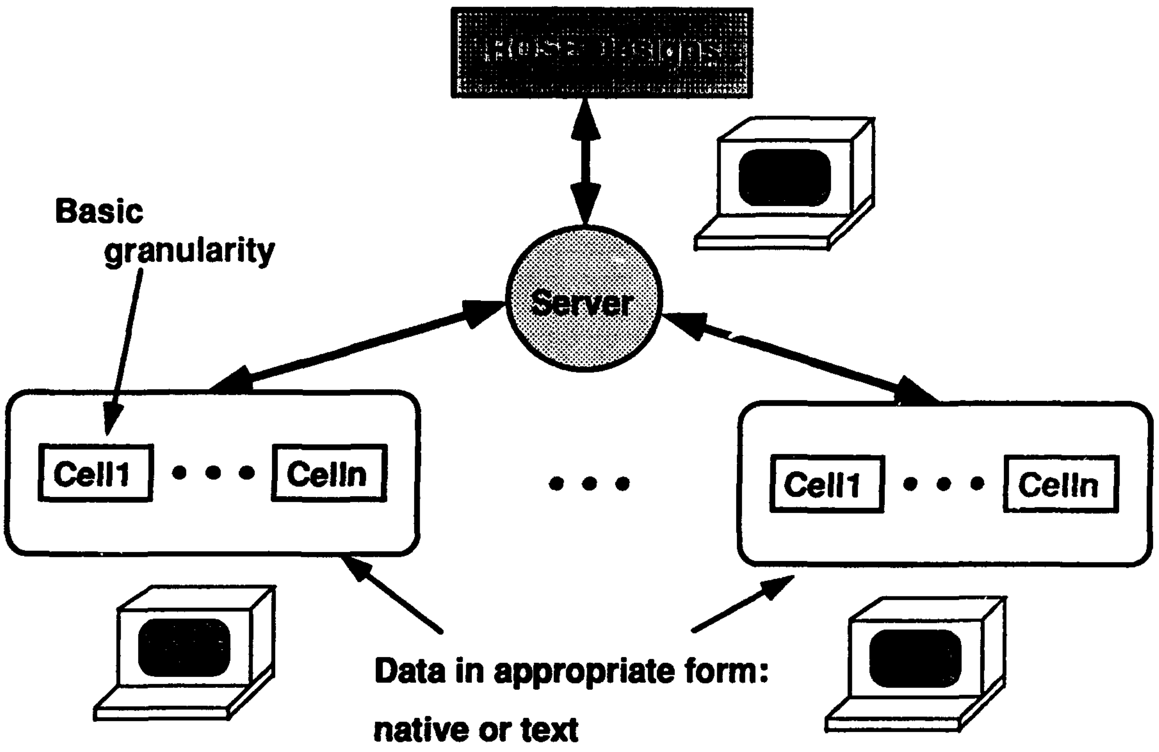

Figure 2. Concurrent Engineering System Operation.

The basic object that is managed by ROSE is the Cell., and any transaction between a user and the system involves at least the transfer of a complete Cell entity. That implies that layout designers using the system should see that data be organized and structured in such a way as to make the transfer of data relatively painless by limiting the individual size of the cells they construct. Good data structuring is desired and a good practice regardless of concurrent engineering requirements, but in our case it is mandatory since N users of the system must be updated whenever one of them makes a change. It must be stressed here that ROSE in and of itself does not impose the restriction to have Cells as basic exchange entities. However, from the practical point of view, if one considers the extreme case of defining the exchange granularity at the level of changing say the location of just a point within a polygon entity plus other such entities, then at that level there would be a large network traffic coming from all the designers as they do their work, caused by relatively minute changes in their own local databases, triggering corresponding ROSE database searches for the objects being modified , added or deleted. Since the natural unit of work in layout is the Cell, it is equally natural to take it as the transaction unit and to have the ROSE data update only when the designer finishes a sizable amount of work, namely , a Cell. This approach also eliminates the need for database searches for individual objects, as the whole cell gets updated every time. However, one must remember the need to keep the Cells slim.

Every Cell entity is stored as an individual ROSE design. The storage is handled in one dedicated workstation with efficient disk access, where a custom server is listening for user requests coming from other workstations in the network, and which when requested will invoke and see that the necessary ROSE-CAD system link software is invoked when the user needs to store or read an update. It will also notify the other concurrent engineering team members whenever there are updates. These users then can invoke the update service at their convenience.

As a user reads an update, a backup copy of his current working design is kept. If a team member finds the update unacceptable for any reason, he must communicate with the other team members as to why that particular update is not good. After discussion and consensus is reached, either he will accept the update and modify whatever else he needs to make it palatable, or else the unacceptable cell will be further modified to meet his requirements. In any case, any designer with write privileges can always reinstate older backup cells back to ROSE, if that is the consensus.

4.4. Mechanism to save and update information

As outlined above, the Cell is the basic transaction unit we envision in our MCM layout concurrent engineering environment. As a designer completes work on a Cell, he saves it to his CAD system database and also notifies the ROSE server, who will receive the data and create the ROSE description of the Cell. Because the server runs in a different workstation than that of the user, the designer will experience a minimum overhead from his saving to the ROSE environment. Since the Cell updates are not occurring frequently the network traffic is minimized that way. Once the Cell has been stored as a ROSE design, the ROSE server will undertake the task to translate it to all the other formats necessary to update the other CAD systems, and will notify the users that an update of that Cell has occurred. When a user wants to update his data from ROSE, he will notify the server to send all the new Cells received since his last update.

4.5. Special considerations

Certain CAD systems have implemented their internal databases with a minimum of hierarchy to them. Essentially, these systems, which have evolved from Printed-Circuit Board layout applications to the MCM domain keep "packages", "pad stacks" and "boards" as basic entities. A board describes the MCM, and contains packages and interconnect . The packages are built out of pad stacks and other geometric and electrical information. These systems are usually very useful to carry out the routing portion of the MCM layout task, but they are not especially suited to handle large amounts of data, notably for the case of certain MCM technologies which customize the MCM substrate starting from a generic, prefabricated part which is personalized in the last stage of manufacturing [DC91]. Such parts are mass produced ahead of time and contain a great deal of interconnect which will end up not used by the personalization step (typical utilization runs up to about 60% of available interconnect density for those substrates). In such cases, it is better to describe the part in full in a hierarchical CAD system. If one wants to use a non-hierarchical CAD system for the routing in a concurrent engineering approach it is convenient to pass only the placement information and the netlist to it, and then to store back into ROSE the interconnect generated. Alternatively, if it proves to be possible to work effectively in the non-hierarchical system with all the MCM data, it would not be advisable to store back into ROSE the full design after routing since that would create a flat and large Cell with all the layout data in it. In any case what one needs is to only store the result of the interconnect (routing) step in a new Cell, to be shared by all the Concurrent Engineering Environment users. Furthermore, if it is possible to carry out the interconnect step itself in a sequence of smaller interconnect tasks (e.g. memory subsystem only, followed by others) it is advisable to store the results of each of those tasks as independent Cells.

Yet another scenario, which is not generally applicable but which covers a large portion of cases, consists in having all members of the design team collaborate in the definition of the basic packages and their placement, and then to submit the interconnect task to the non-hierarchical system (or systems) as the last step of the layout design activity.

In order to implement this program, it is necessary to establish the strategy clearly in the minds of the members of the Concurrent Engineering team. The software side of this process relies on the fact that all objects that have been created by ROSE carry a unique identifier, the OID (Object Identifier), which can usually be stored by the CAD system as a property of the entity received. Therefore all new interconnect generated locally lacks that property and can be recognized and can be distinguished from the previously existing data. Alternatively, it is possible to generate all new interconnect in some special layer, different from that used to store the one already present. In this way, the Cell consisting of just the new interconnect can be created and stored as a new ROSE design and then shared by all other users.

5. Technical Results

5.1. Cadence Edge-ROSE link established

During this reporting period, the principal technical result was that a bi-directional link was established and demonstrated between the Cadence Edge and the ROSE systems. The fundamentals of how this link works have been described above and in particular in the discussion of Figure 1. The MCC information model was written as a schema in the EXPRESS language. This schema was then compiled into C++ classes through the express2c++ software tool from STEP Tools, Inc. [ST]
At the same time, an interface to the internal database of Cadence Edge was developed, making use of the SKILL language from Cadence Design Systems. The role of that interface is to allow the user of Edge to save his work both to the Cadence data format and also into ROSE objects, and also to allow him to read in any updates. The mechanism, as described earlier is to go through an intermediate form. That form is picked up by the other side of the software interface we built. On the way in to the ROSE system, the intermediate form is parsed, on the way out of ROSE the object representation of the Cells are written in the intermediate form.

It should be noted that when the Cadence Edge system reconstructs the Cells from the intermediate form, it requires the procedure to occur from the bottom up, that is, simple Cells must be constructed before building more complex ones. Since every Cell is represented by an individual intermediate file and an individual ROSE design file, it is necessary to process the files in the right order. A special auxiliary file is created which lists the subcells of any given cell in the right order.

The layout schema we have utilized is included in this report as Appendix A. The SKILL software is found in Appendix B.

The parser of the intermediate form into ROSE objects was written using YACC++. A simple lexical analyzer was written in this context. The code to write out ROSE objects into the intermediate form was written in C++ and relies heavily on the ROSE methods to navigate and manage the objects.

The parser can be found as Appendix C and the ROSE-to-intermediate-form code is in Appendix D. All of the appendices included here are for informational purposes only. The software is still subject to change and will continue to change at least through the duration of Phase 1 of this project.

6. Conclusions

6.1. Phase 1 - 1992 Second quarter

We have successfully integrated the first of four MCM layout systems to the ROSE data manager. This is the first step in the implementation of a concurrent engineering environment for the design of MCMs at MCC. We have also established a strategy and a methodology suitable for the concurrent design effort, which we will utilize and validate during the design activity portion of this project.

Also, as part of the validation of the tools , we will be in a position to check the performance and throughput of the ROSE data manager once our heterogeneous set of CAD systems is fully integrated.

References

[DC91] D. Carey et. al. "Variations in MCM Implementations Using a Semi-Custom Technology", Proceedings of the International Electronics Packaging Society (IEPS). 1991, pp. 94-106

[NCGA] IGES documentation can be obtained from the National Computer Graphics Association, IGES/PDES Organization, 2722 Merrilee Dr., Suite 200, Fairfax, VA 22031, (703)698-9600

[ST] "Tutorial Manual for the ROSE++ data manager" and "Reference Manual for the ROSE++ data manager", STEP Tools Inc., Rensselaer Technology Park, Troy, NY, 12180, (518) 276-6751

Appendices

Appendix A
A Layout Information Model, V1.0

```
(*************************************************************************************)
(*                                                                                   *)
(* Copyright 1992, Microelectronics and Computer Technology Corporation *)
(*                    All rights reserved                                           *)
(*                                                                                   *)
(*************************************************************************************)


SCHEMA cadence;

ENTITY Property;
name: STRING;
property: STRING;
selfIdent: STRING;
END_ENTITY;

ENTITY Point
SUBTYPE OF (CadenceObj);
x: REAL;
y: REAL;
END_ENTITY;

ENTITY BBx
SUBTYPE OF (CadenceObj);
ll: Point;
ur: Point;
END_ENTITY;

ENTITY Layer
SUBTYPE OF (CadenceObj);
lyr: INTEGER;
LayerName: STRING;
END_ENTITY;

ENTITY TrueInstance
SUBTYPE OF (CadenceObj);
rep: Cell;
blockName: STRING;
orient: STRING;
xy: Point;
END_ENTITY;

ENTITY Donut
SUBTYPE OF (CadenceObj);
bBox: BBx;
hole: BBx;
lyr: Layer;
shape: STRING;
```

```
END_ENTITY;

ENTITY EllipseArc
SUBTYPE OF (CadenceObj);
bBox: BBx;
ellipsebBox: BBx;
lyr: Layer;
shape: STRING;
END_ENTITY;

ENTITY Label
SUBTYPE OF (CadenceObj);
draftingP: BOOLEAN;
font: STRING;
height: REAL;
justify: STRING;
labelType: STRING;
lyr: Layer;
orient: STRING;
rotatedp: BOOLEAN;
shape: STRING;
theLabel: STRING;
xy: Point;
END_ENTITY;

ENTITY Line
SUBTYPE OF (CadenceObj);
lyr: Layer;
nPath: INTEGER;
path: LIST OF Point;
shape: STRING;
END_ENTITY;

ENTITY Path
SUBTYPE OF (CadenceObj);
beginExt: REAL;
endExt: REAL;
lyr: Layer;
nPath: INTEGER;
path: LIST OF Point;
pathShape: STRING;
shape: STRING;
width: REAL;
END_ENTITY;

ENTITY Polygon
SUBTYPE OF (CadenceObj);
lyr: Layer;
nPath: INTEGER;
path: LIST OF Point;
shape: STRING;
END_ENTITY;

ENTITY Rectangle
SUBTYPE OF (CadenceObj);
bBox: BBx;
lyr: Layer;
```

```
shape: STRING;
END_ENTITY;

ENTITY Tile
SUBTYPE OF (CadenceObj);
index: INTEGER;
orient: STRING;
rep: Cell;
END_ENTITY;

ENTITY Mosaic
SUBTYPE OF (CadenceObj);
bBox: BBx;
numInstances: INTEGER;
instanceList: STRING;
columns: INTEGER;
rows: INTEGER;
name: STRING;
personality: LIST OF Tile;
rep: Cell;
uX: REAL;
uY: REAL;
xy: Point;
orient: STRING;
END_ENTITY;


ENTITY Cell
SUBTYPE OF (CadenceObj);
blockName: STRING;
objList: LIST OF CadenceObj;
repid: STRING;
END_ENTITY;

ENTITY elPin
SUBTYPE OF (CadenceObj);
pinName: STRING;
pinNum: INTEGER;
padRef: TrueInstance;
cellRef: TrueInstance;
END_ENTITY;

ENTITY Net
SUBTYPE OF (CadenceObj);
netName: STRING;
netNum: INTEGER;
elPinList: LIST  OF  elPin;
END_ENTITY;

ENTITY Netlist
SUBTYPE OF (CadenceObj);
netList: LIST OF Net;
END_ENTITY;

ENTITY CadenceObj;
prop: LIST OF Property;
selfIdent: STRING;
```

```
Obid: STRING;
cellType: STRING;
END_ENTITY;

END_SCHEMA;
```

Appendix B
SKILL programs, V1.0

```
;*****************************************************************************************/
;*                                                                                       */
;* Copyright 1992, Microelectronics and Computer Technology Corporation */
;*                    All rights reserved                                              */
;*                                                                                       */
;*****************************************************************************************/
```

```
;-------------------------------------------------------------------------
; Totcds2txt.il
; -------------------------------------------------------------------------
procedure(totcds2txt()
   rep = getWindowRep()
   level = 0
   mastersVisited = list()
   filename = nameGen(rep)
   homedir = getShellEnvVar("HOME")
   dir = strcat(homedir "/design." filename)
   system("mkdir %s" dir)
   totauthor(rep filename)
   postorder(rep filename)
)


procedure(pwrite(filename)
         homedir = getShellEnvVar("HOME")
         dir = strcat(homedir "/design." filename)
         parsefile = strcat(dir "/" filename)
         parser = getShellEnvVar("ROSE_PARSE")
         printf("%s %s " parser parsefile)
         debug2 = system("%s %s" parser parsefile)
         printf("%d " debug2)
)


procedure(nameGen(rep)
   prog((name trepname tblockname)
         name = rep~>fullPathName
         delimiter = "/"
         len = strlen(name)
         revision = snipString(name delimiter)
         revlen = 1 + strlen(revision)
         x = len - revlen
         trepname = substring(name 1 x)
         repname = snipString(trepname delimiter)
         replen = 1 + strlen(repname)
         y = len - replen - revlen
         tblockname = substring(name 1 y)
         blockname = snipString(tblockname delimiter)
```

```
          tempfile = strcat(blockname delimiter repname delimiter revision)
          return(blockname)
   )
)

procedure(snipString(string delimiter)
   prog((temp snipped)
          temp = rindex(string delimiter)
          snipped = substring(temp 2)
          return(snipped)
   )
)

procedure(postorder(rep tname)
      outname = strcat(dir "/" tname ".tree")
      file = outfile(outname)
      headerInfo(rep)
      cellList(rep 1 "true" 0 0 )
      close(file)
      sortedoutname = strcat(outname ".postorder")
      system("sort -d -r -u -o %s %s" sortedoutname outname)
)

procedure(headerInfo(rep)
      printf("*** Hierarchy of Design:  %s %s\n" rep~>blockName rep~>repName)
      fprintf(file "*** Hierarchy of Design:  %s %s\n" rep~>blockName rep~>repName)
      foreach(i rep~>prop
         if(i~>name == "userUnits" then
            printf("*** User Units : %s \n" i~>value)
            fprintf(file "*** User Units  : %s \n" i~>value)
         )
         if(i~>name == "graphicsEditorUnits per userUnit" then
            printf("*** GraphicsEditorUnits per User Unit  : %f \n" i~>value)
            fprintf(file "*** GraphicsEditorUnits per User Unit  : %f \n" i~>value)
         )
      )
      printf("\n")
      fprintf(file "\n")
)


/*
The procedure cellList is a recursive routine that does a hierarchical
traversal of the design. The parameter "absolute" should have the value
"true" to make all coordinates relative to the top-level cell. The
parameter "absolute" should have the value "false" to make all the
coordinates relative to the next higher level cell.
*/

procedure(cellList(rep level absolute x y )
   prog( (tmprep errorMessage name repList)
      printMessage("Hierarchy Traversal in Progress: Looking at Cell: %s" rep~>blockName)
      foreach(inst rep~>instances
         if( (inst~>purpose == "cell") && ( inst~>type == "trueInst") then
                  fprintf(file "%d" level)
                  printf("%d" level)
          for(i 0 level
             fprintf(file "  ")
             printf("  ")
          )
          name = inst~>master~>blockName
          bName = inst~>blockName
          rot = inst~>orient
          printf("(Lev_%d) %s \n" level bName)
          fprintf(file "(Lev_%d) %s \n" level bName)
                 master = inst~>master
                 if(!member(master mastersVisited) then
                          totauthor(master bName)
                          mastersVisited = cons(master mastersVisited)
                 )
          repList = list("layout" "symbolic")
          found = "false"
          while( car(repList) != nil && found != "true"
             sprintf(cell "%s %s" name car(repList))
             tmprep = dbOpen( cell)
             if( tmprep != nil then found = "true")
             repList = cdr(repList)
          )
          if(found != "true" then
             sprintf( errorMessage "Unable to open '%s'." name)
             println(errorMessage)
          else
             if(tmprep~>instances != nil then
                   level = level + 1
                   if(absolute == "true" then

                                     new_x = x + xCoord(car(inst~>bBox))
                                     new_y = y + yCoord(car(inst~>bBox))
                      cellList(tmprep level "true" new_x new_y)
                   else
                      cellList(tmprep level "false" 0 0)
                   )
                   level = level - 1
             )
             dbClose(tmprep)
          )
       )
       if((inst~>purpose == "cell") && (inst~>type == "mosaicSubInst") then
          fprintf(file "%d" level)
          printf("%d" level)
          for(i 0 level
             fprintf(file "  ")
             printf("  ")
          )
          name = inst~>master~>blockName
          bName = inst~>blockName
          rot = inst~>orient
                 if((rot == nil) then
                    rot = "none")
          printf("(Lev_%d) %s \n" level bName)
          fprintf(file "(Lev_%d) %s \n" level bName)
          master = inst~>master
          if(!member(master mastersVisited) then
                    totauthor(master bName)
                    mastersVisited = cons(master mastersVisited)
                )
                repList = list("layout" "symbolic")
                found = "false"
                while( car(repList) != nil && found != "true"
                   sprintf(cell "%s %s" name car(repList))
                   tmprep = dbOpen( cell)
                   if( tmprep != nil then found = "true")
                   repList = cdr(repList)
                )
                if(found != "true" then
                   sprintf( errorMessage "Unable to open '%s'." name)
                   println(errorMessage)
                else
                   if(tmprep~>instances != nil then
                         level = level + 1
                         if(absolute == "true" then

                            new_x = x + xCoord(car(inst~>bBox))
                            new_y = y + yCoord(car(inst~>bBox))
                            cellList(tmprep level "true" new_x new_y)
                         else
                            cellList(tmprep level "false" 0 0)
                         )
                         level = level - 1
                   )
                   dbClose(tmprep)
                )
             )
          )
       )
    )


procedure(totauthor(rep tempname)
   prog((port type newfilename)
         newfilename = strcat(dir "/" tempname)
         port=outfile(newfilename)
         if(((rep~>OID) == nil) then
            dbCreateProp(rep "OID" "string" "new"))
         fprintf(port "Cellname: \t")
         fprintf(port "%s\n" tempname)
         fprintf(port "RepID: \t")
         println(rep port)
         fprintf(port "OID \t")
         fprintf(port "%s\n" (rep~>OID))
         foreach(el rep~>shapes
            if(((el~>OID) == nil) then
                  addoid(rep))

            type=(el~>shape)

            if((type=="rectangle") then
                  fprintf(port "shape \t")
                  fprintf(port "%s\n" (el~>shape))
                  fprintf(port "bBox \t")
          list2coords(port (el~>bBox))
          fprintf(port "layer \t")
println((el~>layer) port)
          fprintf(port "OID \t")
fprintf(port "%s\n" (el~>OID) port))

 if((type=="polygon") then
          fprintf(port "shape \t")
          fprintf(port "%s\n" (el~>shape))
          fprintf(port "nPath \t")
println((el~>nPath) port)
          fprintf(port "path \t")
          list2coords(port (el~>path))
          fprintf(port "layer \t")
println((el~>layer) port)
          fprintf(port "OID \t")
          fprintf(port "%s\n" (el~>OID) port))

 if((type=="donut") then
          fprintf(port "shape \t")
          fprintf(port "%s\n" (el~>shape))
          fprintf(port "bBox \t")
          list2coords(port (el~>bBox))
          fprintf(port "hole \t")
          list2coords(port (el~>hole))
          fprintf(port "layer \t")
println((el~>layer) port)
          fprintf(port "OID \t")
          fprintf(port "%s\n" (el~>OID) port))

 if((type=="ellipse") then
          fprintf(port "shape \t")
          fprintf(port "%s\n" (el~>shape))
          fprintf(port "ellipseBBox \t' )
          if(!((el~>ellipseBBox) == nil) then
                    list2coords(port (el~>ellipseBBox))
             else
                    println((el~>ellipseBBox) port)
          )
          fprintf(port "bBox \t")
          list2coords(port (el~>bBox))
          fprintf(port "layer \t")
println((el~>layer) port)
          fprintf(port "OID \t")
          fprintf(port "%s\n" (el~>OID) port))

 if((type=="label") then
          fprintf(port "shape \t")
          fprintf(port "%s\n" (el~>shape))
          fprintf(port "draftingP \t")
fprintf(port "%s\n" (el~>draftingP))
          fprintf(port "font \t")
fprintf(port "%s\n" (el~>font))
          fprintf(port "height \t")
          println((el~>height) port)
          fprintf(port "justify \t")
fprintf(port "%s\n" (el~>justify))
```

```
            fprintf(port "labelType \t")
    fprintf(port "%s\n" (el~>labelType))
            fprintf(port "orient \t")
            fprintf(port "%s\n" (el~>orient))
            fprintf(port "theLabel \t")
    fprintf(port "%s\n" (el~>theLabel))
            fprintf(port "xy \t")
            fprintf(port " %f %f \n" car(el~>xy) cadr(el~>xy))
            fprintf(port "layer \t")
    println((el~>layer) port)
            fprintf(port "OID \t")
            fprintf(port "%s\n" (el~>OID) port))

     if((type=="line") then
            fprintf(port "shape \t")
            fprintf(port "%s\n" (el~>shape))
    fprintf(port "nPath \t")
    println((el~>nPath) port)
    fprintf(port "path \t")
            list2coords(port (el~>path))
    fprintf(port "layer \t")
    println((el~>layer) port)
            fprintf(port "OID \t")
            fprintf(port "%s\n" (el~>OID) port))

     if((type=="path") then
            fprintf(port "shape \t")
            fprintf(port "%s\n" (el~>shape))
            fprintf(port "nPath \t")
    println((el~>nPath) port)
            fprintf(port "path \t")
            list2coords(port (el~>path))
            fprintf(port "pathShape \t")
    fprintf(port "%s\n" (el~>pathShape))
            fprintf(port "width \t")
    println((el~>width) port)
            fprintf(port "layer \t")
    println((el~>layer) port)
            fprintf(port "OID \t")
            fprintf(port "%s\n" (el~>OID) port))
   )
  foreach(el rep~>instances
     if((el~>purpose == "cell") && (el~>type == "trueInst") then
            if(((el~>OID) == nil) then
               addoid(rep))
            fprintf(port "cell \t")
            fprintf(port "%s\n" (el~>blockName))
            fprintf(port "xy \t")
            fprintf(port " %f %f \n" car(el~>xy) cadr(el~>xy))
            fprintf(port "orient \t")
            fprintf(port "%s\n" (el~>orient))
            fprintf(port "OID \t")
            fprintf(port "%s\n" (el~>OID) port))

     if((el~>purpose == "cell") && (el~>type == "mosaicSubInst") then
            if(((el~>OID) == nil) then
        addoid(rep))
```

```
            fprintf(port "array \t")
                  fprintf(port "%s\n" (el~>blockName))
            fprintf(port "xy \t")
                  fprintf(port " %f %f \n" car(el~>mosaic~>xy) cadr(el~>mosaic~>xy))
                  fprintf(port "columns \t")
                  println((el~>mosaic~>columns) port)
            fprintf(port "rows \t")
            println((el~>mosaic~>rows) port)
                  fprintf(port "uX \t")
                  println((el~>mosaic~>uX) port)
                  fprintf(port "uY \t")
                  println((el~>mosaic~>uY) port)
            fprintf(port "orient \t")
            fprintf(port "%s\n" (car(el~>mosaic~>personality)~>orient))
            fprintf(port "OID \t")
                  fprintf(port "%s\n" (el~>OID) port))
                  acname = nameGen(el~>master)
                  totauthor(el~>master acname)
          )
          fprintf(port "END FILE\n")
          close(port)
          parsefile = strcat("design." filename "/" tempname)
          system("parse++ %s" parsefile)
      )
)


procedure(addoid(cellmaster)
   tcellname=strcat((cellmaster~>blockName) " layout " "current")
   tcell=dbOpen(tcellname nil "a")
   foreach(obj cellmaster~>shapes
          if(((obj~>OID) == nil) then
             dbCreateProp(obj "OID" "string" "new"))
   )
   foreach(obj cellmaster~>instances
      if(((obj~>OID) == nil) then
         dbCreateProp(obj "OID" "string" "new"))
   )
   dbSave(tcell)
   dbClose(tcell)
)


procedure(list2coords(theport alist)
          foreach(piece alist
                  Xcoord = car(piece)
                  Ycoord = cadr(piece)
                  fprintf(theport " %f %f " Xcoord Ycoord))
          fprintf(theport "\n")
)
;-----------------------------------------------------------------------
; Modcds2txt.il
; ----------------------------------------------------------------------
procedure(modcds2txt()
   rep = getWindowRep()
   level = 0
   mastersVisited = list()
   filename = modnameGen(rep)
   homedir = getShellEnvVar("HOME")
   dir = strcat(homedir "/design." filename)
   system("mkdir %s" dir)
   modauthor(rep filename)
   modpostorder(rep filename)
   parsefile = strcat(dir "/" filename)
   parser = getShellEnvVar("ROSE_PARSE")
   system("%s %s" parser parsefile)
)


procedure(modnameGen(rep)
   prog((name trepname tblockname)
         name = rep~>fullPathName
         delimiter = "/"
         len = strlen(name)
         revision = modsnipString(name delimiter)
         revlen = 1 + strlen(revision)
         x = len - revlen
         trepname = substring(name 1 x)
         repname = modsnipString(trepname delimiter)
         replen = 1 + strlen(repname)
         y = len - replen - revlen
         tblockname = substring(name 1 y)
         blockname = modsnipString(tblockname delimiter)
         tempfile = strcat(blockname delimiter repname delimiter revision)
         return(blockname)
   )
)

procedure(modsnipString(string delimiter)
   prog((temp snipped)
         temp = rindex(string delimiter)
         snipped = substring(temp 2)
         return(snipped)
   )
)

procedure(modpostorder(rep tname)
      outname = strcat(dir "/" tname ".tree")
      file = outfile(outname)
      modheaderInfo(rep)
      modcellList(rep 1 "true" 0 0 )
      close(file)
      sortedoutname = strcat(outname ".postorder")
      system("sort -d -r -u -o %s %s" sortedoutname outname)
)

procedure(modheaderInfo(rep)
      printf("** Hierarchy of Design:  %s %s\n" rep~>blockName rep~>repName)
      fprintf(file "** Hierarchy of Design:  %s %s\n" rep~>blockName rep~>repName)
      foreach(i rep~>prop
         if(i~>name == "userUnits" then
            printf("** User Units : %s \n" i~>value)
            fprintf(file "** User Units  : %s \n" i~>value)
```

```
        )
        if(i~>name == "graphicsEditorUnits per userUnit" then
           printf("** GraphicsEditorUnits per User Unit  : %f \n" i~>value)
           fprintf(file "*** GraphicsEditorUnits per User Unit  : %f \n" i~>value)
        )
     )
     printf("\n")
     fprintf(file "\n")
)
```

```
/*
The procedure modcellList is a recursive routine that does a hierarchical
traversal of the design. The parameter "absolute" should have the value
"true" to make all coordinates relative to the top-level cell. The
parameter "absolute" should have the value "false" to make all the
coordinates relative to the next higher level cell.
*/

procedure(modcellList(rep level absolute x y )
   prog( (tmprep errorMessage name repList)
      printMessage("Hierarchy Traversal in Progress: Looking at Cell: %s" rep~>blockName)
      foreach(inst rep~>instances
         if( (inst~>purpose == "cell") && ( inst~>type == "trueInst") then
                  fprintf(file "%d" level)
                  printf("%d" level)
            for(i 0 level
               fprintf(file "  ")
               printf("  ")
            )
            name = inst~>master~>blockName
            bName = inst~>blockName
            rot = inst~>orient
            printf("(Lev_%d) %s \n" level bName)
            fprintf(file "(Lev_%d) %s \n" level bName)
                  master = inst~>master
                  if(!member(master mastersVisited) then
                           modauthor(master bName)
                           mastersVisited = cons(master mastersVisited)
                  )
            repList = list("layout" "symbolic")
            found = "false"
            while( car(repList) != nil && found != "true"
               sprintf(cell "%s %s" name car(repList))
               tmprep = dbOpen( cell)
               if( tmprep != nil then found = "true")
               repList = cdr(repList)
            )
            if(found != "true" then
               sprintf( errorMessage "Unable to open '%s'." name)
               println(errorMessage)
            else
               if(tmprep~>instances != nil then
                     level = level + 1
                     if(absolute == "true" then
                        new_x = x + xCoord(car(inst~>bBox))
                        new_y = y + yCoord(car(inst~>bBox))
                  modcellList(tmprep level "true" new_x new_y)
                else
                  modcellList(tmprep level "false" 0 0)
                )
                level = level - 1
            )
            dbClose(tmprep)
          )
        )
        if((inst~>purpose == "cell") && (inst~>type == "mosaicSubInst") then
          fprintf(file "%d" level)
          printf("%d" level)
          for(i 0 level
            fprintf(file "  ")
            printf("  ")
          )
          name = inst~>master~>blockName
          bName = inst~>blockName
          rot = inst~>orient
                if((rot == nil) then
                   rot = "none")
          printf("(Lev_%d) %s \n" level bName)
          fprintf(file "(Lev_%d) %s \n" level bName)
          master = inst~>master
          if(!member(master mastersVisited) then
               modauthor(master bName)
               mastersVisited = cons(master mastersVisited)
          )
          repList = list("layout" "symbolic")
          found = "false"
          while( car(repList) != nil && found != "true"
            sprintf(cell "%s %s" name car(repList))
            tmprep = dbOpen( cell)
            if( tmprep != nil then found = "true")
            repList = cdr(repList)
          )
          if(found != "true" then
            sprintf( errorMessage "Unable to open '%s'." name)
            println(errorMessage)
          else
            if(tmprep~>instances != nil then
                level = level + 1
                if(absolute == "true" then

                  new_x = x + xCoord(car(inst~>bBox))
                  new_y = y + yCoord(car(inst~>bBox))
                  modcellList(tmprep level "true" new_x new_y)
                else
                  modcellList(tmprep level "false" 0 0)
                )
                level = level - 1
            )
            dbClose(tmprep)
          )
        )
        )
      )
    )

procedure(modauthor(rep tempname)
  prog((port type filename)
        if(((rep~>modifiedButNotSavedP) == "FALSE") then
          return(nil)
        )
        filename = strcat(dir "/" tempname)
        port=outfile(filename)
        if(((rep~>OID) == nil) then
          dbCreateProp(rep "OID" "string" "new"))
        fprintf(port "Cellname: \t")
        fprintf(port "%s\n" tempname)
        fprintf(port "RepID: \t")
        println(rep port)
        fprintf(port "OID \t")
        fprintf(port "%s\n" (rep~>OID))
        foreach(el rep~>shapes
          if(((el~>OID) == nil) then
                modaddoid(rep))

          type=(el~>shape)

          if((type=="rectangle") then
                fprintf(port "shape \t")
                fprintf(port "%s\n" (el~>shape))
                fprintf(port "bBox \t")
                modlist2coords(port (el~>bBox))
                fprintf(port "layer \t")
          println((el~>layer) port)
                fprintf(port "OID \t")
          fprintf(port "%s\n" (el~>OID) port))

          if((type=="polygon") then
                fprintf(port "shape \t")
                fprintf(port "%s\n" (el~>shape))
                fprintf(port "nPath \t")
          println((el~>nPath) port)
                fprintf(port "path \t")
                modlist2coords(port (el~>path))
                fprintf(port "layer \t")
          println((el~>layer) port)
                fprintf(port "OID \t")
                fprintf(port "%s\n" (el~>OID) port))

          if((type=="donut") then
                fprintf(port "shape \t")
                fprintf(port "%s\n" (el~>shape))
                fprintf(port "bBox \t")
                modlist2coords(port (el~>bBox))
                fprintf(port "hole \t")
                modlist2coords(port (el~>hole))
                fprintf(port "layer \t")
          println((el~>layer) port)
        fprintf(port "OID \t")
        fprintf(port "%s\n" (el~>OID) port))

  if((type=="ellipse") then
        fprintf(port "shape \t")
        fprintf(port "%s\n" (el~>shape))
        fprintf(port "ellipseBBox \t")
        if(!((el~>ellipseBBox) == nil) then
                modlist2coords(port (el~>ellipseBBox))
          else
                println((el~>ellipseBBox) port)
        )
        fprintf(port "bBox \t")
        modlist2coords(port (el~>bBox))
        fprintf(port "layer \t")
 println((el~>layer) port)
        fprintf(port "OID \t")
        fprintf(port "%s\n" (el~>OID) port))

  if((type=="label") then
        fprintf(port "shape \t")
        fprintf(port "%s\n" (el~>shape))
        fprintf(port "draftingP \t")
 fprintf(port "%s\n" (el~>draftingP))
        fprintf(port "font \t")
 fprintf(port "%s\n" (el~>font))
        fprintf(port "height \t")
        println((el~>height) port)
        fprintf(port "justify \t")
 fprintf(port "%s\n" (el~>justify))
        fprintf(port "labelType \t")
 fprintf(port "%s\n" (el~>labelType))
        fprintf(port "orient \t")
        fprintf(port "%s\n" (el~>orient))
        fprintf(port "theLabel \t")
 fprintf(port "%s\n" (el~>theLabel))
        fprintf(port "xy \t")
        fprintf(port " %f %f \n" car(el~>xy) cadr(el~>xy))
        fprintf(port "layer \t")
 println((el~>layer) port)
        fprintf(port "OID \t")
        fprintf(port "%s\n" (el~>OID) port))

  if((type=="line") then
        fprintf(port "shape \t")
        fprintf(port "%s\n" (el~>shape))
 fprintf(port "nPath \t")
 println((el~>nPath) port)
 fprintf(port "path \t")
        modlist2coords(port (el~>path))
 fprintf(port "layer \t")
 println((el~>layer) port)
        fprintf(port "OID \t")
        fprintf(port "%s\n" (el~>OID) port))

  if((type=="path") then
        fprintf(port "shape \t")
```

```
              fprintf(port "%s\n" (el~>shape))
              fprintf(port "nPath \t")
       println((el~>nPath) port)
              fprintf(port "path \t")
              modlist2coords(port (el~>path))
              fprintf(port "pathShape \t")
       fprintf(port "%s\n" (el~>pathShape))
              fprintf(port "width \t")
       println((el~>width) port)
              fprintf(port "layer \t")
       println((el~>layer) port)
              fprintf(port "OID \t")
              fprintf(port "%s\n" (el~>OID) port))
    )
    foreach(el rep~>instances
       if((el~>purpose == "cell") && (el~>type == "trueInst") then
              if(((el~>OID) == nil) then
                 modaddoid(rep))
              fprintf(port "cell \t")
              fprintf(port "%s\n" (el~>blockName))
              fprintf(port "xy \t")
              fprintf(port " %f %f \n" car(el~>xy) cadr(el~>xy))
              fprintf(port "orient \t")
              fprintf(port "%s\n" (el~>orient))
              fprintf(port "OID \t")
              fprintf(port "%s\n" (el~>OID) port))

       if((el~>purpose == "cell") && (el~>type == "mosaicSubInst") then
              if(((el~>OID) == nil) then
           modaddoid(rep))
       fprintf(port "array \t")
              fprintf(port "%s\n" (el~>blockName))
       fprintf(port "xy \t")
              fprintf(port " %f %f \n" car(el~>mosaic~>xy) cadr(el~>mosaic~>xy))
              fprintf(port "columns \t")
              println((el~>mosaic~>columns) port)
       fprintf(port "rows \t")
       println((el~>mosaic~>rows) port)
              fprintf(port "uX \t")
              println((el~>mosaic~>uX) port)
              fprintf(port "uY \t")
              println((el~>mosaic~>uY) port)
       fprintf(port "orient \t")
       fprintf(port "%s\n" (car(el~>mosaic~>personality)~>orient))
       fprintf(port "OID \t")
              fprintf(port "%s\n" (el~>OID) port))
              acname = modnameGen(el~>master)
              modauthor(el~>master acname)
    )
    fprintf(port "END FILE\n")
    close(port)
  )
)

procedure(modaddoid(cellmaster)
  tcellname=strcat((cellmaster~>blockName) " layout " "current")
  tcell=dbOpen(tcellname nil "a")
  foreach(obj cellmaster~>shapes
        if(((obj~>OID) == nil) then
           dbCreateProp(obj "OID" "string" "new"))
  )
  foreach(obj cellmaster~>instances
     if(((obj~>OID) == nil) then
        dbCreateProp(obj "OID" "string" "new"))
  )
  dbSave(tcell)
  dbClose(tcell)
)

procedure(modlist2coords(theport alist)
        foreach(piece alist
                Xcoord = car(piece)
                Ycoord = cadr(piece)
                fprintf(theport " %f %f " Xcoord Ycoord))
        fprintf(theport "\n")
)
```

Appendix C

C++ parser, V1.0

```
/*****************************************************************************************/
/*                                                                                      */
/* Copyright 1992, Microelectronics and Computer Technology Corporation */
/*                  All rights reserved                                          */
/*                                                                                      */
/*****************************************************************************************/
%{
#include "rose.h"
#include "cadence.h"
#include <string.h>
#include <stdio.h>
/* declare (List,Point);
declare (List,CadenceObj);
declare (List,Property);
declare (List,Tile); */
/* implement (List,Point);
implement (List,CadenceObj);
implement (List,Property);
implement (List,Tile); */

typedef struct xyp {double x; double y;} *XYPAIR;
struct xyp  xypair1,xypair2;

%}

%union {
    double num; // for doubles (REAL)
        char * name_ptr; // for STRING
        int  val; // for int  (INTEGER)
        XYPAIR xypr; // for xy pairs
        BBx * bbx; // for BBx
        Layer * lay; // for layers
        Rectangle * rect; // for Rectangles
    EllipseArc * ell; // For EllipseArcs
        Donut * donut; // for Donuts
        ListOfPoint * traj; // for Lists of points
    Path * pat; // for paths
        Polygon * poly; // for polygons
        Label * lab; // for labels
        Line * line; // for lines
        TrueInstance * trueins; // for true instances
        Mosaic * mosaic; // for Mosaics or arrays
        Cell * cell; // for cells
}

%token <val> END
%token <val> CELLNAME
%token <val> REPID
%token <val> OID
%token <val> SHAPE
%token <val> ELLIPSEBBOX
```

```
%token <val> BBOX
%token <val> LAYER
%token <val> HOLE
%token <val> NPATH
%token <val> PATH
%token <val> SHAPEPATHSHAPE
%token <val> WIDTH
%token <val> DRAFTINGP
%token <val> JUSTIFICATION
%token <val> ORIENTATION
%token <val> LABELTYPE
%token <val> THELABEL
%token <val> FONT
%token <val> HEIGHT
%token <val> XY
%token <val> CELL
%token <val> ARRAY
%token <val> ROWS
%token <val> COLUMNS
%token <val> DELTAX
%token <val> DELTAY

%token <name_ptr> ELLIPSESHAPE
%token <name_ptr> RECTANGLESHAPE
%token <name_ptr> POLYGONSHAPE
%token <name_ptr> PATHSHAPE
%token <name_ptr> LINESHAPE
%token <name_ptr> LABELSHAPE
%token <name_ptr> NIL
%token <name_ptr> DONUTSHAPE

%token <val> NUMBER
%token <num> FNUMBER
%token <name_ptr> STRING

%type <xypr> xypair
%type <bbx> bBox
%type <bbx> ellipsebbox
%type <bbx> hole
%type <rect> rectangledescriptor
%type <rect> rectangle
%type <lay> layer
%type <ell> ellipsedescriptor
%type <ell> ellipse
%type <donut> donutdescriptor
%type <donut> donut
%type <val> npath
%type <traj> trajectory
%type <pat> path
%type <pat> pathdescriptor
%type <num> width
%type <name_ptr> pathshape
%type <poly> polygon
%type <poly> polygondescriptor
%type <lab> labeldescriptor
%type <lab> label
%type <val> draftingstatus
```

```
%type <name_ptr> font
%type <num> height
%type <name_ptr> justification
%type <name_ptr> labeltype
%type <name_ptr> orientation
%type <name_ptr> theLabel
%type <xypr> xy
%type <line> line
%type <line> linedescriptor
%type <trueins> cellinstance
%type <val> columns
%type <val> rows
%type <num> deltax
%type <num> deltay
%type <mosaic> arraydescriptor
%type <mosaic> array
%type <name_ptr> repid
%type <name_ptr> oid
%type <cell> cellcontents
%type <cell> cell
/* %type <pattern> devicedescriptor */ /* if you want to return from stack */
%%

text_file: /* nothing */
| text_file cell end

 end : END STRING


 cell : CELLNAME  STRING repid oid cellcontents { $5->blockName($2);
                                                   $5->repid($3);
                                                   $5->Obid($4);
//                                         cout << form("Oid = %s\n",$5->Obid());
                                                  $$ = $5;
                                                 }


 repid :  REPID  STRING { $$ = $2; }

 oid : OID  STRING {  $$ = $2; }

 xypair: FNUMBER FNUMBER {
                           xypair1.x = $1;
                           xypair1.y = $2;
                           $$ =  &xypair1; }

 cellcontents : /* nothing */ { Cell * cellptr;
                                cellptr = pnew Cell();
                                ListOfCadenceObj * lcoptr;
                                lcoptr = pnew ListOfCadenceObj();
                                cellptr->objList(lcoptr);
                                $$ = cellptr;
                              }
 | cellcontents ellipse oid  { ListOfCadenceObj * lcoptr;
//                                   cout << form("Oid = %s\n",$3);
                              $2->Obid($3);
                              lcoptr = $1->objList();
```

```
                                        lcoptr->add($2);
                                    }

    | cellcontents rectangle oid { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents donut oid    { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents path oid     { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents label oid    { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents line oid     { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents polygon oid  { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents cellinstance oid  { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }
    | cellcontents array oid  { ListOfCadenceObj * lcoptr;
                                        $2->Obid($3);
//                                              cout << form("Oid = %s\n",$2->Obid());
                                        lcoptr = $1->objList();
                                        lcoptr->add($2);
                                    }


    ellipse : SHAPE ELLIPSESHAPE ellipsedescriptor
            { $3->shape($2);
              $$ = $3;
            }

ellipsedescriptor : ellipsebbox bBox layer
            {
                    EllipseArc * ellptr;
                    ellptr = pnew EllipseArc();
                    ellptr->lyr($3);
                    ellptr->ellipsebBox($1);
                    ellptr->bBox($2);
                    $$ = ellptr;
                }


ellipsebbox : ELLIPSEBBOX NIL
        {
                BBx *bbxptr = NULL;
                $$ = bbxptr;
            }
| ELLIPSEBBOX xypair {xypair2.x = $2->x; xypair2.y = $2->y;} xypair
      {
                BBx *bbxptr;
                bbxptr = pnew BBx();
                Point *pt1,*pt2;
                pt1 = pnew Point();
                pt2 = pnew Point();
                pt1->x(xypair2.x);
                pt1->y( xypair2.y);
                pt2->x($4->x);
                pt2->y($4->y);
                bbxptr->ll(pt1);
                bbxptr->ur(pt2);
                $$ = bbxptr;
            }

bBox : BBOX xypair {xypair2.x = $2->x; xypair2.y = $2->y;} xypair
      {
                BBx *bbxptr;
                bbxptr = pnew BBx();
                Point *pt1,*pt2;
                pt1 = pnew Point();
                pt2 = pnew Point();
                pt1->x(xypair2.x);
                pt1->y( xypair2.y);
                pt2->x($4->x);
                pt2->y($4->y);
                bbxptr->ll(pt1);
                bbxptr->ur(pt2);
                $$ = bbxptr;
            }


layer : LAYER NUMBER {
                Layer *lyrptr;
                        lyrptr = pnew Layer();
                        lyrptr->lyr($2);
                        $$ = lyrptr;
```

```
                                        }

rectangle : SHAPE RECTANGLESHAPE rectangledescriptor { $3->shape($2);
                                        $$ = $3;
                                                    }


rectangledescriptor : bBox layer
                { Rectangle *recptr;
                        recptr = pnew Rectangle();
                        recptr->bBox($1);
                        recptr->lyr($2);
                        $$ = recptr;
                      }

donut : SHAPE DONUTSHAPE donutdescriptor  { $3->shape($2);
                                $$ = $3;
                                        }

donutdescriptor : bBox hole layer                    {
                        Donut * donptr;
                        donptr = pnew Donut();
                        donptr->lyr($3);
                        donptr->bBox($1);
                        donptr->hole($2);
                        $$ = donptr;
                    }

hole : HOLE xypair {xypair2.x = $2->x; xypair2.y = $2->y;} xypair
        {
                BBx *bbxptr;
                bbxptr = pnew BBx();
                Point *pt1,*pt2;
                pt1 = pnew Point();
                pt2 = pnew Point();
                pt1->x(xypair2.x);
                pt1->y( xypair2.y);
                pt2->x($4->x);
                pt2->y($4->y);
                bbxptr->ll(pt1);
                bbxptr->ur(pt2);
                $$ = bbxptr;
            }

path : SHAPE PATHSHAPE pathdescriptor { $3->shape($2);
                                        $$ = $3; }

pathdescriptor : npath trajectory pathshape width layer
            { Path * pathptr;
                    pathptr = pnew Path();
                    pathptr->nPath($1);
                    pathptr->path($2);
                    pathptr->pathShape($3);
                    pathptr->width($4);
                    pathptr->lyr($5);
                    $$ = pathptr;
            }
```

```
npath : NPATH NUMBER {$$ = $2;}

trajectory : PATH xypair {xypair2.x = $2->x; xypair2.y = $2->y;} xypair
        {ListOfPoint * lopptr;
                Point *p1,*p2;
                lopptr = pnew ListOfPoint();
                p2 = pnew Point();
                p2->x(xypair2.x);
                p2->y(xypair2.y);
                p1 = pnew Point();
                p1->x(xypair1.x);
                p1->y(xypair1.y);
                lopptr->add(p2);
                lopptr->add(p1);
                $$ = lopptr;
                }
| trajectory xypair {Point *pptr;
                    pptr = pnew Point();
                    pptr->x(xypair1.x);
                    pptr->y(xypair1.y);
                    $1->add(pptr);
                    $$ = $1;
                  }

pathshape : SHAPEPATHSHAPE STRING {$$ = $2;}

width : WIDTH FNUMBER {$$ = $2;}

polygon : SHAPE POLYGONSHAPE polygondescriptor
        { $3->shape($2);
                $$ = $3;
            }

polygondescriptor : npath trajectory layer
            { Polygon * polyptr;
                    polyptr = pnew Polygon();
                    polyptr->nPath($1);
                    polyptr->path($2);
                    polyptr->lyr($3);
                    $$ = polyptr;
             }

label : SHAPE LABELSHAPE labeldescriptor { $3->shape($2); $$ = $3; }

labeldescriptor : draftingstatus font height justification labeltype orientation theLabel xy layer
            { Label * lblptr;
                    lblptr = pnew Label();
                    lblptr->draftingP($1);
                    lblptr->font($2);
                    lblptr->height($3);
                    lblptr->justify($4);
                    lblptr->labelType($5);
                    lblptr->orient($6);
                    lblptr->theLabel($7);
                    Point * pptr;
                    pptr = pnew Point();
                    pptr->x($8->x);
                    pptr->y($8->y);
                    lblptr->xy(pptr);
                    lblptr->lyr($9);
                    $$ = lblptr;
                   }


draftingstatus : DRAFTINGP STRING { if(strcmp($2,"TRUE") == 0) $$ = TRUE;
                                     else $$ = FALSE;}

font : FONT STRING  { $$ = $2; }

height : HEIGHT FNUMBER  { $$ = $2; }

justification : JUSTIFICATION STRING { $$ = $2; }

labeltype : LABELTYPE STRING { $$ = $2; }

orientation : ORIENTATION NUMBER { char orient[50];
                                    int integ = $2;
                                    //orient << form("%s",(char *)(&integ));
                                    // istream itoken;
                                    // itoken.istream(&integ);
                                    // itoken >> orient;
                                    // strcpy(orient,"0");
                                    /* sprintf(orient,"%s",$2); */
                              // $$ = orient;
                                    if(integ == 0) strcpy(orient,"0");
                                    if(integ == 90) strcpy(orient,"90");
                                    if(integ == 180) strcpy(orient,"180");
                                    if(integ == 270) strcpy(orient,"270");
                                    $$ = orient;
                                   }
   | ORIENTATION STRING { $$ = $2; }


theLabel : THELABEL STRING {$$ = $2;}
| theLabel STRING  { int l1,l2;
                     l1 = strlen($1);
                     l2 = strlen($2);
                     char * cptr;
                     cptr = malloc(l1+l2+2);
                     strcpy(cptr,$1);

                     strcat(cptr," ");
                     strcat(cptr,$2);
                     $$ = cptr;}

xy : XY xypair  {$$ = $2;}

line : SHAPE LINESHAPE linedescriptor  {$3->shape($2); $$ = $3; }

linedescriptor : npath trajectory layer
        { Line * lineptr;
              lineptr = pnew Line();
              lineptr->nPath($1);
```

```
                    lineptr->path($2);
                    lineptr->lyr($3);
                    $$ = lineptr;
            }


 cellinstance : CELL STRING xy orientation  { TrueInstance * tinsptr;
                                               tinsptr = pnew TrueInstance();
                                               tinsptr->blockName($2);
                                               Point * pptr;
                                               pptr = pnew Point();
                                               pptr->x($3->x);
                                               pptr->y($3->y);
                                               tinsptr->xy(pptr);
                                               tinsptr->orient($4);
                                               $$ = tinsptr;
                                             }

 array : ARRAY STRING arraydescriptor { $3->name($2); $$ = $3; }

 arraydescriptor : xy columns rows deltax deltay orientation
                { Mosaic * mosptr;
                        mosptr = pnew Mosaic();
                        Point * pptr;
                        pptr = pnew Point();
                        pptr->x($1->x);
                        pptr->y($1->y);
                        mosptr->xy(pptr);
                        mosptr->columns($2);
                        mosptr->rows($3);
                        mosptr->uX($4);
                        mosptr->uY($5);
                        mosptr->orient($6);
                        $$ = mosptr;
                    }

 columns : COLUMNS NUMBER  { $$ = $2; }

 rows : ROWS NUMBER  { $$ = $2; }

 deltax : DELTAX FNUMBER { $$ = $2; }

 deltay : DELTAY FNUMBER { $$ = $2; }
%%

#include <ctype.h>
#include <stream.h>
#include <string.h>

char *progname;
filebuf fileb;
istream file(&fileb);
int lineno = 1;
int eol=1;

void main(int argc, char **argv)
{
  progname = argv[0];
  if (argc == 2)
        {

            if(fileb.open(argv[1],input) == 0)
  {
  cout << form("Sorry, I couldn't open %s for parsing\n",argv[1]);
  exit(1);
   }
        cerr << "Starting Rose design" << flush;

            ROSE.newDesign(argv[1]);

            cerr << "Starting yyparse\n";

                        yyparse();
//          cerr << "Displaying.\n";

//      ROSE.display();
            cerr << " Saving\n";
        ROSE.saveDesign();

                }

        else yyerror("Usage: parse++ input_file \n");
}
void warning(char *s, char *t)
{
  cout << form("%s; %s",progname, s);
        if(t)
                cout << form(" %s",t);
        cout << form(" near line %d\n",lineno);
}


void yyerror(char *s)
{
        warning(s,(char *) 0);
}

void echo(char *s)
{
        cout << form("%s ",s);
}

char *getToken(istream ifile)
{
 static char buf[1024];
 ifile >> buf;
 if(!ifile) {         cerr << "Null read from input\n";
                      return(NULL);
              }

 return buf;
}
yylex()
{
        char c;
        char string[50];
        char *ptr;

//      cout << form("Value of eol: %d\n",eol);
        if(file.eof()) return 0;

        file.get(c);
        if( c == '\n') { lineno++;
                         eol = 1;
                         }
        while(c == ' ' || c == '\t' || c == '\n') {if(file.eof()) return 0;
                                                   file.get(c);
                                                   if( c == '\n') { lineno++;
                                                                    eol = 1;
                                                                    }
                                                   }
        file.putback(c);
        {int i=0;
         while(c != ' ' && c != '\t' && c != '\n') {if(file.eof()) return 0;
                                                    file.get(c);
                                                   string[i++] = c;

                                                                    }
         file.putback(c);
         string[i-1] = '\0';
         c = string[i-2];
        }
//      file >> string;
        int len = strlen(string);

//      cout << form("Just read: %s\n",string);
//      cout << form("The last character is %s\n",string + len -1);
//      cout << form("The value of c is: %s\n",&c);

        if (eol && c != '\n')
        {
                /* keyword */
//        cout << "Decided it is a keyword\n";
          eol = 0;
           ptr = (char *)malloc(strlen(string) + 1);
                if(ptr == NULL)
                {
                        printf("Error at yylex() allocating memory for a string\n");
                        exit(0);
                }
                                                        {
                                                                int i;
                                                                for(i=0; i<(strlen(string) + 1);i++) *(ptr +
i) = '\0';
                                                                }
                strcpy(ptr,string);
//            cout << form("ptr contains the string: %s\n",ptr);
                yylval.name_ptr = ptr;
                if(strcmp(ptr,"END") == 0) return END;
                if(strcmp(ptr,"Cellname:") == 0) { // cout << "Passed through 1\n";
                                                    return CELLNAME; }
```

```
                    if(strcmp(ptr,"RepID:") == 0) { // yyerror("Passed through 2\n");
                                                 return REPID;
                                            }
                      if(strcmp(ptr,"OID") == 0) return OID;
                    if(strcmp(ptr,"shape") == 0) return SHAPE;
                      if(strcmp(ptr,"ellipseBBox") == 0) return ELLIPSEBBOX;
                    if(strcmp(ptr,"bBox") == 0) return BBOX;
                    if(strcmp(ptr,"hole") == 0) return HOLE;
                    if(strcmp(ptr,"nPath") == 0) return NPATH;
                    if(strcmp(ptr,"path") == 0) return PATH;
                    if(strcmp(ptr,"pathShape") == 0) return SHAPEPATHSHAPE;
                    if(strcmp(ptr,"width") == 0) return WIDTH;
                    if(strcmp(ptr,"draftingP") == 0) return DRAFTINGP;
                    if(strcmp(ptr,"justify") == 0) return JUSTIFICATION;
                    if(strcmp(ptr,"orient") == 0) return ORIENTATION;
                    if(strcmp(ptr,"labelType") == 0) return LABELTYPE;
                    if(strcmp(ptr,"theLabel") == 0) return THELABEL;
                    if(strcmp(ptr,"xy") == 0) return XY;
                    if(strcmp(ptr,"font") == 0) return FONT;
                    if(strcmp(ptr,"height") == 0) return HEIGHT;
                    if(strcmp(ptr,"cell") == 0) return CELL;
                    if(strcmp(ptr,"array") == 0) return ARRAY;
                    if(strcmp(ptr,"columns") == 0) return COLUMNS;
                    if(strcmp(ptr,"rows") == 0) return ROWS;
                    if(strcmp(ptr,"uX") == 0) return DELTAX;
                    if(strcmp(ptr,"uY") == 0) return DELTAY;
                    if(strcmp(ptr,"layer") == 0) return LAYER;


                    return STRING;
               }
               if (isdigit((int)c) || c == '.' || c == '-')
               {
                    /* number */
//             cout << "Decided it is a number\n";
               {int j, dflag = 0, dateflag = 0;
                for(j=0;j<strlen(string);j++)
                { if(isalpha(string[j])) {  ptr = (char *)malloc(strlen(string) + 1);
                                                            if(ptr == NULL)
                                                            {
                                                                 printf("Error in yylex ()
allocating memory for a string\n");
                                                                 exit(0);
                                                            }
                                                       {
                                                            int i;
                                                            for(i=0; i<(strlen(string) + 1);i++) *(ptr +
i) = '\0';
                                                            }
                                                            strcpy (ptr,string);
                                                            yylval.name_ptr = ptr;
                                                            return STRING;
                                                       }
                   if(!isdigit(string[j]) ) {dflag = 1;
                                                                              if(string[j] ==
(char)':') /* date */
```

```
                                                                          dateflag
= 1;
                                                             }
 }
        if(dflag == 1 && dateflag == 0)
        { yylval.num =  atof(string);
          return FNUMBER;
 }
        else
        {  if (dateflag == 0)
                {
                        yylval.val = atoi(string);
                        return NUMBER;
                }
        else
        {
                ptr = (char *)malloc(strlen(string) + 1);
                if(ptr == NULL)
                {
                        printf("Error in yylex () allocating memory for a string\n");
                        exit(0);
                }
                                                {
                                                        int i;
                                                        for(i=0; i<(strlen(string) + 1);i++) *(ptr +
i) = '\0';
                                                        }
      strcpy (ptr,string);
                yylval.name_ptr = ptr;
                return STRING;
        }
                }
}
        }
        if (isalpha((int)c))
        {
                /* string */
//       cout << "decided it is alpha\n";
                ptr = (char *)malloc(strlen(string) + 1);
                if(ptr == NULL)
                {
                        printf("Error in yylex() allocating memory for a string\n");
                        exit(0);
                }
                                                {
                                                        int i;
                                                        for(i=0; i<(strlen(string) + 1);i++) *(ptr +
i) = '\0';
                                                        }
                strcpy(ptr,string);
//              cout << form("ptr contains the string: %s\n",ptr);
                yylval.name_ptr = ptr;

                if(strcmp(ptr,"ellipse") == 0) return ELLIPSESHAPE;
                if(strcmp(ptr,"rectangle") == 0) return RECTANGLESHAPE;
                if(strcmp(ptr,"polygon") == 0) return POLYGONSHAPE;
                if(strcmp(ptr,"path") == 0) return PATHSHAPE;
                if(strcmp(ptr,"line") == 0) return LINESHAPE;
                    if(strcmp(ptr,"label") == 0) return LABELSHAPE;
                    if(strcmp(ptr,"nil") == 0) return NIL;
                    if(strcmp(ptr,"donut") == 0) return DONUTSHAPE;


                    return STRING;}
        return c;
}
```

Appendix D
ROSE to EDGE, V1.0

```
/************************************************************************************/
/*                                                                                  */
/* Copyright 1992, Microelectronics and Computer Technology Corporation */
/*                 All rights reserved                                        */
/*                                                                                  */
/************************************************************************************/
#include "rose.h"
// #include "RoseOID.h"
#include "cadence.h"
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <stream.h>
#include <string.h>

// BOOL cvtOIDtoSTR(OID oid, STR str);

char *progname;
filebuf fileb;
ostream file(&fileb);
filebuf treefileb;
ostream treefile(&treefileb);
char oidd[43];
char dirname[100];
char filname[100];
//////////////////////////////////////////////////////////////////////
//
// Ellipse
//
//////////////////////////////////////////////////////////////////////

void write_ellipsearc(EllipseArc * elarc)

{
                                        file << form("shape ellipse\n");
                                        if(elarc->ellipsebBox()) {
                                         file << form("ellipseBBox %f %f %f %f\n",elarc-
>ellipsebBox()->ll()->x(),elarc->ellipsebBox()->ll()->y(),elarc->ellipsebBox()->ur()->x(),elarc-
>ellipsebBox()->ur()->y());}
                                        else
                                         file << form("ellipseBBox  nil\n");

                                        if (elarc->bBox())
                                          {
                                         file << form("bBox %f %f %f %f\n",elarc->bBox()->ll()-
>x(),elarc->bBox()->ll()->y(),elarc->bBox()->ur()->x(),elarc->bBox()->ur()->y());}
                                        else
                                         file << form("bBox  nil\n");
                                        file << form("layer  %d\n",elarc->lyr()->lyr());
                                        ROSE.index()->cvtOIDtoSTR(elarc->oid(),oidd);
                                        file << form("OID   %s\n",oidd);
}

////////////////////////////////////////////////////////////////////
//
// Donut
//
////////////////////////////////////////////////////////////////////

void write_donut(Donut * donut)
{
 file << form("shape   donut\n");
 if(donut->bBox()) {
                         file << form("bBox %f %f %f %f\n",donut->bBox()->ll()->x(),donut->bBox()->ll()-
>y(),donut->bBox()->ur()->x(),donut->bBox()->ur()->y());}
                                        else
                                         file << form("bBox  nil\n");
 if(donut->hole()) {
                         file << form("hole %f %f %f %f\n",donut->hole()->ll()->x(),donut->hole()->ll()-
>y(),donut->hole()->ur()->x(),donut->hole()->ur()->y());}
                                        else
                                         file << form("hole  nil\n");
 file << form("layer   %d\n",donut->lyr()->lyr());

                                        ROSE.index()->cvtOIDtoSTR(donut->oid(),oidd);
                                        file << form("OID   %s\n",oidd);
}
////////////////////////////////////////////////////////////////////
//
// Polygon
//
////////////////////////////////////////////////////////////////////
void write_polygon(Polygon * polygon)
{
 file << form("shape   polygon\n");

 file << form("nPath   %d\n",polygon->nPath());

 file << form("path    ");
 int k;
 ListOfPoint *list = polygon->path();
 for(k=0; k< polygon->nPath(); k++)
   { file << form("%f %f  ",(*list)[k]->x(),(*list)[k]->y()); }

 file << form("\n");

 file << form("layer   %d\n",polygon->lyr()->lyr());

 ROSE.index()->cvtOIDtoSTR(polygon->oid(),oidd);
 file << form("OID   %s\n",oidd);

}
////////////////////////////////////////////////////////////////////
//
// Rectangle
//
```

```
/////////////////////////////////////////////////////////////////////
void write_rectangle(Rectangle * rectangle)
{
 file << form("shape   rectangle\n");
 file << form("bBox   %f  %f  %f  %f\n",rectangle->bBox()->ll()->x(),
                              rectangle->bBox()->ll()->y(),
                              rectangle->bBox()->ur()->x(),
                              rectangle->bBox()->ur()->y());
 file << form("layer  %d\n",rectangle->lyr()->lyr());
 ROSE.index()->cvtOIDtoSTR(rectangle->oid(),oidd);
 file << form("OID   %s\n",oidd);

}
/////////////////////////////////////////////////////////////////////
//
// Path
//
/////////////////////////////////////////////////////////////////////
void write_path(Path * path)
{
 file << form("shape   path\n");
 file << form("nPath   %d\n",path->nPath());

 file << form("path    ");
 int k;
 ListOfPoint *list = path->path();
 for(k=0; k< path->nPath(); k++)
   { file << form("%f  %f  ",(*list)[k]->x(),(*list)[k]->y()); }

 file << form("\n");

 file << form("pathShape   %s\n",path->pathShape());
 file << form("width  %f\n",path->width());
 file << form("layer  %d\n",path->lyr()->lyr());
 ROSE.index()->cvtOIDtoSTR(path->oid(),oidd);
 file << form("OID   %s\n",oidd);

}
/////////////////////////////////////////////////////////////////////
//
// Label
//
/////////////////////////////////////////////////////////////////////
void write_label(Label * label)
{
 file << form("shape   label\n");
 if(label->draftingP()) { file << form("draftingP  TRUE\n");}
 else
   {file << form("draftingP   FALSE\n");}

 file << form("font  %s\n",label->font());
 file << form("height   %f\n",label->height());
 file << form("justify  %s\n",label->justify());
 file << form("labelType  %s\n",label->labelType());
 file << form("orient  %d\n",atoi(label->orient()));
 file << form("theLabel  %s\n",label->theLabel());
 file << form("xy  %f %f\n",label->xy()->x(),label->xy()->y());
 file << form("layer  %d\n",label->lyr()->lyr());
 ROSE.index()->cvtOIDtoSTR(label->oid(),oidd);
 file << form("OID   %s\n",oidd);

}
/////////////////////////////////////////////////////////////////////
//
// Line
//
/////////////////////////////////////////////////////////////////////
void write_line(Line * line)
{
 file << form("shape   line\n");
 file << form("nPath   %d\n",line->nPath());

 file << form("path    ");
 int k;
 ListOfPoint *list = line->path();
 for(k=0; k< line->nPath(); k++)
   { file << form("%f  %f  ",(*list)[k]->x(),(*list)[k]->y()); }

 file << form("\n");
 file << form("layer  %d\n",line->lyr()->lyr());
 ROSE.index()->cvtOIDtoSTR(line->oid(),oidd);
 file << form("OID   %s\n",oidd);

}
/////////////////////////////////////////////////////////////////////
//
// write_trueinstance
//
/////////////////////////////////////////////////////////////////////
void write_trueinstance(TrueInstance * trueinstance)
{
 file << form("cell   %s\n",trueinstance->blockName());
 file << form("xy  %f  %f\n",trueinstance->xy()->x(),trueinstance->xy()->y());
 file << form("orient  %d\n",atoi(trueinstance->orient()));
 ROSE.index()->cvtOIDtoSTR(trueinstance->oid(),oidd);
 file << form("OID   %s\n",oidd);

}
/////////////////////////////////////////////////////////////////////
//
// Mosaic
//
/////////////////////////////////////////////////////////////////////
void write_mosaic(Mosaic * mosaic)
{
 file << form("array   %s\n",mosaic->name());
 file << form("xy  %f  %f\n",mosaic->xy()->x(),mosaic->xy()->y());
 file << form("columns  %d\n",mosaic->columns());
 file << form("rows  %d\n",mosaic->rows());
 file << form("uX  %f\n",mosaic->uX());
 file << form("uY  %f\n",mosaic->uY());
 file << form("orient  %d\n",atoi(mosaic->orient()));
 ROSE.index()->cvtOIDtoSTR(mosaic->oid(),oidd);
 file << form("OID   %s\n",oidd);
```

```
}
///////////////////////////////////////////////////////////////////
//
// write_tree
//
///////////////////////////////////////////////////////////////////
void write_tree(List (CadenceObj) aList, int depth)

{

          int n = aList.size();

          for(int i =0; i< n; i++)
           {
              if(aList[i]->isa("TrueInstance"))
               {
                char localname[200];
                for(int m=0; m<200; m++) localname[m] = '\0';

                 TrueInstance * trueinstance = ROSE_CAST(TrueInstance, aList[i]);
                 treefile << form("%d (Lev_%d) %s\n",depth+1,depth+1,trueinstance->blockName());
                strcat(localname,dirname);
                strcat(localname,"/");
                strcat(localname,trueinstance->blockName());
                 RoseDesign * design =   ROSE.useDesign(localname);

                 List (CadenceObj) bList;

                 ROSE.findObjects(&bList);
                 int k = bList.size();
                 for(int j =0; j<k; j++)
                  {
                       if((bList[j]->isa("TrueInstance")) ||(bList[j]->isa("Mosaic")))
                        {
                         write_tree(bList,depth+1);
                        }
                  }
               }

              if(aList[i]->isa("Mosaic"))
               {
                char localname[200];
                for(int m=0; m<200; m++) localname[m] = '\0';
                 Mosaic * mosaic = ROSE_CAST(Mosaic, aList[i]);
                 treefile << form("%d (Lev_%d) %s\n",depth+1,depth+1,mosaic->name());
                strcat(localname,dirname);
                strcat(localname,"/");
                strcat(localname,mosaic->name());

                 RoseDesign * design =   ROSE.useDesign(localname);

                 List (CadenceObj) bList;

                 ROSE.findObjects(&bList);
                 int k = bList.size();
                 for(int j =0; j<k; j++)
```

```
                    {
                        if((bList[j]->isa("TrueInstance")) || (bList[j]->isa("Mosaic")))
                         {
                          write_tree(bList,depth+1);
                         }
                    }
                 }
             }
          }
////////////////////////////////////////////////////////////////////
//
// main
//
////////////////////////////////////////////////////////////////////

void main(int argc, char **argv)
{

   progname = argv[0];
   if (argc == 3)
         {
            strcpy(dirname,argv[1]);
            strcpy(filname,argv[2]);
            char fullname[200];
         strcat(fullname,dirname);
            strcat(fullname,"/");
         strcat(fullname,filname);
            if(fileb.open(fullname,output) == 0)
   {
   cout << form("Sorry, I couldn't open %s for output\n",argv[1]);
   exit(1);
    }
         cerr << "Reading Rose design" << flush;

            RoseDesign * design = ROSE.useDesign(fullname);
//          CadenceObj * topcell = ROSE_CAST(CadenceObj, design->root() );
            List (CadenceObj) aList;

            ROSE.findObjects(&aList);

            int n = aList.size();
            char treefilename[100];
            strcpy(treefilename,dirname);
            strcat(treefilename,"/");
         strcat(treefilename,filname);
            strcat(treefilename,".tree.postorder");

            for(int i =0; i< n; i++)
             {

                  if(aList[i]->isa("Cell"))
                    {
                     Cell * cell = ROSE_CAST(Cell, aList[i]);
                     file << form("Cellname:   %s\n",cell->blockName());
                     file << form("RepID:   %s\n",cell->repid());
                     ROSE.index()->cvtOIDtoSTR(cell->oid(),oidd);
                     file << form("OID   %s\n",oidd);
```

```
                ListOfCadenceObj *list = cell->objList();
                int siz = list->size();
                int k;
                for(k=0; k< siz; k++)
                                {
                                 if((*list)[k]->isa("EllipseArc"))
                                  {EllipseArc * elarc = ROSE_CAST(EllipseArc,(*list)[k]);
                                  write_ellipsearc(elarc); }
                                 if((*list)[k]->isa("Donut"))
                                  { Donut * donut = ROSE_CAST(Donut,(*list)[k]);
                                      write_donut(donut); }
                                 if((*list)[k]->isa("Polygon"))
                                  {Polygon * polygon = ROSE_CAST(Polygon,(*list)[k]);
                                      write_polygon(polygon); }
                                 if((*list)[k]->isa("Rectangle"))
                                  {Rectangle * rectangle = ROSE_CAST(Rectangle,(*list)[k]);
                                      write_rectangle(rectangle); }
                                 if((*list)[k]->isa("Path"))
                                  {Path * path = ROSE_CAST(Path,(*list)[k]);
                                      write_path(path); }
                                 if((*list)[k]->isa("Label"))
                                  {Label * label = ROSE_CAST(Label,(*list)[k]);
                                      write_label(label); }
                                 if((*list)[k]->isa("Line"))
                                  {Line * line = ROSE_CAST(Line,(*list)[k]);
                                      write_line(line); }
                                 if((*list)[k]->isa("TrueInstance"))
                                  {TrueInstance * trueinstance = ROSE_CAST(TrueInstance,(*list)[k]);
                                      write_trueinstance(trueinstance); }
                                 if((*list)[k]->isa("Mosaic"))
                                  {Mosaic * mosaic = ROSE_CAST(Mosaic,(*list)[k]);
                                      write_mosaic(mosaic); }


                                }
                 file << form("END FILE\n");
        if(treefileb.open(treefilename,output) == 0)
         {
              cout << form("Sorry, I couldn't open %s for output\n",treefilename);
              exit(2);
         }
                int depth = 0;
              write_tree(aList,depth);
              treefile << form("**\n");
                treefileb.close();

                               }


              }
              system(form("sort -d -u -r -o %s %s\n",treefilename,treefilename));
         }

      else cerr << form("Usage: r2c designdirname designname \n");
}
```

Acknowledgments

Team Members

The efforts of Shaune Stark from MCC are acknowledged. He has written the SKILL procedures to interface to the Cadence EDGE database and contributed to the definition of the architecture and operation of the system.
Chuck Gannon from Harris Electric Design Automation and Deborah Cobb from MCC have collaborated in the definition of the connectivity portion of the layout information model.
Christopher Hudnall from MCC has been helpful with all matters related to the installation of ROSE at MCC, as well as UNIX support.

External support

It is a pleasure to acknowledge the support of Martin Hardwick, Alok Mehta and in general of the employees of STEP Tools, Inc., in all matters regarding ROSE.